Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

Claim 1. (currently amended) A network management system comprising:

~~a plurality of plug-in mapping modules, wherein the plurality of plug-in mapping modules are each operable to provide one or more mappings for managed object data types between an~~ a plurality of plug-in mapping modules, wherein each of the plurality of plug-in mapping modules is operable to provide a unique mapping for a set of managed object data types between the same two languages for describing data associated with managed objects, wherein one of the two languages is an interface definition language (IDL) and the other of the two languages is an abstract syntax notation, wherein the interface definition language comprises a language for defining interfaces to managed objects across a plurality of platforms and across a plurality of programming languages, wherein the managed objects comprise instances of the managed object data types, and wherein the abstract syntax notation comprises a language for describing data; and

a mapping framework, wherein the mapping framework is operable to receive the plurality of plug-in mapping modules, and wherein the mapping framework is operable to provide access to the plurality of plug-in mapping modules to facilitate the mapping of managed object data types in accordance with the mappings for managed object data types provided by the plurality of mapping modules.

Claim 2. (original) The system of claim 1, wherein the managed objects comprise a telephone system.

Claim 3. (original) The system of claim 1, wherein the managed objects comprise a network switch.

Claim 4. (original) The system of claim 1, wherein the mapping framework comprises a plurality of processes which are concurrently executable.

Claim 5. (original) The system of claim 1, wherein the interface definition language is operable to provide a single mapping which is applicable to any managed object class.

Claim 6. (original) The system of claim 1, wherein the abstract syntax notation comprises Abstract Syntax Notation One (ASN1).

Claim 7. (original) The system of claim 1,

wherein the mapping framework comprises a converter framework library, wherein the converter framework library comprises a set of abstract classes which provide an interface for the one or more plug-in mapping modules, and wherein the interface comprises wrappers to a plurality of corresponding converter implementation classes; and

wherein a converter implementation library comprises the one or more plug-in mapping modules, wherein the one or more plug-in mapping modules comprise the plurality of converter implementation classes, and wherein the converter implementation classes provide mappings for the managed object data types between the interface definition language and the abstract syntax notation.

Claim 8. (original) The system of claim 7, wherein the converter framework library comprises an ASN1 converter framework library, and wherein the ASN1 converter framework library provides an interface to map managed object data types between ASN1 and the interface definition language.

Claim 9. (original) The system of claim 7, wherein the converter implementation library comprises a C++ IDL to ASN1 converter implementation library, and wherein the C++ IDL to ASN1 converter implementation library comprises C++ classes and functions to map managed object data types between ASN1 and the interface definition language.

Claim 10. (original) The system of claim 7, wherein the converter framework library comprises a collection of classes, and wherein the classes comprise wrappers to corresponding implementation classes in the converter implementation library.

Claim 11. (currently amended) A method for managing a network, the method comprising:

selecting a first mapping module from a plurality of plug-in mapping modules, wherein each of the plurality of mapping modules is operable to provide a unique mapping between a first set of data types and a second set of data types;

inputting a first data type from ~~a~~ the first set of data types, wherein the first set of data types is expressed in an abstract syntax notation, and wherein the abstract syntax notation comprises a language for describing data;

determining a corresponding second data type from ~~a~~ the second set of data types according to a first mapping provided by a first mapping module, wherein the second set of data types is expressed in an interface definition language (IDL), wherein the interface definition language comprises a language for defining interfaces to managed objects across a plurality of

platforms and across a plurality of programming languages, wherein the interface definition language is class independent, and wherein a mapping framework is operable to receive a plurality of plug-in mapping modules including the first mapping module, wherein the plurality of plug-in mapping modules are operable to provide a plurality of mappings for managed object data types between the interface definition language and the abstract syntax notation, and wherein the managed objects comprise instances of the managed object data types; and

returning the second data type.

Claim 12. (original) The method of claim 11, wherein the mapping framework comprises an abstract converter implementation class, wherein the abstract converter implementation class comprises a superclass for a type-specific converter implementation class in each of the plurality of plug-in mapping modules, wherein the type-specific converter implementation classes are operable to convert specific managed object data types between the interface definition language and the abstract syntax notation.

Claim 13. (original) The method of claim 12, wherein the abstract converter implementation class comprises an abstract ASN1 converter implementation class, wherein the abstract ASN1 converter implementation class comprises an interface class, wherein the interface class comprises a super-class of one or more ASN1 converter implementation classes which are operable to map managed object data types between ASN1 and the interface definition language.

Claim 14. (original) The method of claim 11, wherein the mapping framework comprises an abstract value class, wherein the abstract value class comprises a superclass for a generic value class in each of the plurality of plug-in mapping modules, wherein the generic value classes are operable to hold values in an abstract syntax notation generic type.

Claim 15. (original) The method of claim 14, wherein the generic value classes comprise an IDL value class, wherein the IDL value class is operable to hold IDL values in an abstract syntax notation generic type.

Claim 16. (original) The method of claim 11, wherein the mapping framework comprises an abstract generic converter helper class, wherein the abstract generic converter helper class comprises a superclass for a generic converter helper implementation class in each of the plurality of plug-in mapping modules, wherein the generic converter helper implementation classes are operable to create actual converters based on data types.

Claim 17. (original) The method of claim 16, wherein the abstract generic converter helper class comprises an abstract generic ASN1 converter helper class, wherein the abstract generic ASN1 converter helper class comprises a super-class of one or more ASN1 converter implementation helper classes which are operable to create actual converters based on ASN1 Types.

Claim 18. (currently amended) The method of claim ~~11~~ 16, wherein each of the plurality of plug-in mapping modules comprises a generic converter helper implementation class, wherein the generic converter helper implementation class is a subclass of the abstract generic converter helper class, wherein the generic converter helper implementation class is operable to create actual converters based on data types.

Claim 19. (currently amended) A carrier medium comprising program instructions for managing a network, wherein the program instructions are computer-executable to implement:

selecting a first mapping module from a plurality of plug-in mapping modules, wherein each of the plurality of mapping modules is operable to provide a unique mapping between a first set of data types and a second set of data types;

inputting a first data type from a the first set of data types, wherein the first set of data types is expressed in an abstract syntax notation, and wherein the abstract syntax notation comprises a language for describing data;

determining a corresponding second data type from a the second set of data types according to a first mapping provided by a first mapping module, wherein the second set of data types is expressed in an interface definition language (IDL), wherein the interface definition language comprises a language for defining interfaces to managed objects across a plurality of platforms and across a plurality of programming languages, wherein the interface definition language is class independent, and wherein a mapping framework is operable to receive a plurality of plug-in mapping modules including the first mapping module, wherein the plurality of plug-in mapping modules are operable to provide a plurality of mappings for managed object data types between the interface definition language and the abstract syntax notation, and wherein the managed objects comprise instances of the managed object data types; and

returning the second data type.

Claim 20. (original) The carrier medium of claim 19, wherein the mapping framework comprises an abstract converter implementation class, wherein the abstract converter implementation class comprises a superclass for a type-specific converter implementation class in each of the plurality of plug-in mapping modules, wherein the type-specific converter implementation classes are operable to convert specific managed object data types between the interface definition language and the abstract syntax notation.

Claim 21. (original) The carrier medium of claim 20, wherein the abstract converter implementation class comprises an abstract ASN1 converter implementation class, wherein the abstract ASN1 converter implementation class comprises an interface class,

wherein the interface class comprises a super-class of one or more ASN1 converter implementation classes which are operable to map managed object data types between ASN1 and the interface definition language.

Claim 22. (original) The carrier medium of claim 19, wherein the mapping framework comprises an abstract value class, wherein the abstract value class comprises a superclass for a generic value class in each of the plurality of plug-in mapping modules, wherein the generic value classes are operable to hold values in an abstract syntax notation generic type.

Claim 23. (original) The carrier medium of claim 22, wherein the generic value classes comprise an IDL value class, wherein the IDL value class is operable to hold IDL values in an abstract syntax notation generic type.

Claim 24. (original) The carrier medium of claim 19, wherein the mapping framework comprises an abstract generic converter helper class, wherein the abstract generic converter helper class comprises a superclass for a generic converter helper implementation class in each of the plurality of plug-in mapping modules, wherein the generic converter helper implementation classes are operable to create actual converters based on data types.

Claim 25. (original) The carrier medium of claim 24, wherein the abstract generic converter helper class comprises an abstract generic ASN1 converter helper class, wherein the abstract generic ASN1 converter helper class comprises a super-class of one or more ASN1 converter implementation helper classes which are operable to create actual converters based on ASN1 Types.

Claim 26. (currently amended) The carrier medium of claim ~~19~~ 24, wherein each of the plurality of plug-in mapping modules comprises a generic converter helper implementation class, wherein the generic converter helper implementation class is a

subclass of the abstract generic converter helper class, wherein the generic converter helper implementation class is operable to create actual converters based on data types.

Claim 27. (currently amended) A method for network management, comprising:

providing a mapping framework which comprises one or more abstract classes;

implementing ~~one or more plug-in mapping modules by creating one or more implementation classes as subclasses of the one or more abstract classes in the mapping framework, wherein the plug-in mapping modules are configured to provide mappings for managed object data types between an~~ a plurality of plug-in mapping modules, wherein each of the plurality of mapping modules is operable to provide a unique mapping for a set of managed object metadata types between an interface definition language (IDL) and an abstract syntax notation, wherein the managed objects comprise instances of the managed object data types, wherein the abstract syntax notation comprises a language for describing data, wherein the interface definition language comprises a language for defining interfaces to managed objects across a plurality of platforms and across a plurality of programming languages, and wherein the interface definition language is class independent; and

converting managed object data types between an interface definition language and an abstract syntax notation using one of the plug-in mapping modules.

Claim 28. (original) The method of claim 27, wherein the abstract classes comprise an abstract converter implementation class, wherein creating the one or more implementation classes as subclasses of the one or more abstract classes in the mapping framework further comprises creating a type-specific converter implementation class as a subclass of the abstract converter implementation class, and wherein the type-specific

converter implementation class is operable to convert a managed object data type between the interface definition language and the abstract syntax notation

Claim 29. (original) The method of claim 27, wherein the abstract classes comprise an abstract value class, wherein creating the one or more implementation classes as subclasses of the one or more abstract classes in the mapping framework further comprises creating a generic value class as a subclass of the abstract value class, and wherein the generic value class is operable to hold interface definition language values in an abstract syntax notation generic type.

Claim 30. (original) The method of claim 27, wherein the abstract classes comprise an abstract generic converter helper class, wherein creating the one or more implementation classes as subclasses of the one or more abstract classes in the mapping framework further comprises creating a generic converter helper implementation class as a subclass of the abstract generic converter helper class, and wherein the generic converter helper implementation class is operable to create actual converters based on abstract syntax notation data types.